A Theory for Software-Hardware Co-Scheduling for ASIPs and Embedded Processors

R. Govindarajan

Supercomputer Edn. & Res. Centre
Indian Institute of Science
Bangalore, 560 012, India.

govind@serc.iisc.ernet.in

Erik R. Altman

IBM T.J. Watson Research Centre
P.O.Box 704
Yorktown Heights, NY 10598. U.S.A.

erik@watson.ibm.com

Guang R. Gao

Electrical & Computer Engineering
University of Delaware
Newark, DE 19716, U.S.A.

ggao@capsl.udel.edu

Abstract

Exploiting instruction-level parallelism (ILP) is extremely important for achieving high performance in application specific instruction set processors (ASIPs) and embedded processors. Existing techniques deal with either scheduling hardware pipelines to obtain higher throughput or software pipeline — an instruction scheduling technique for iterative computation — loops for exploiting greater ILP. We integrate these techniques to co-schedule hardware and software pipelines to achieve greater instruction throughput.

In this paper, we develop the underlying theory of co-scheduling, called the *Modulo-Scheduled Pipeline* (or MS-Pipeline) theory. More specifically, we establish the necessary and sufficient condition for achieving the maximum throughput in a given pipeline operating under modulo scheduling. Further, we establish a sufficient condition to achieve a specified throughput, based on which we also develop a methodology for designing the hardware pipelines that achieve such a throughput.

1 Introduction

In order to achieve high performance, Application Specific Instruction set Processors (ASIPs) and embedded processors must exploit Instruction-Level Parallelism (ILP). Modern DSP processors such as TI's C6x [18], Philip's TriMedia [12], and their embedded cores are very long instruction word (VLIW) architectures to exploit ILP. Unlike conventional general purpose processors, ASIPs and embedded processors are customized for a specific class of applications and hence their architectures are optimized extensively through expensive design methodology.

In addition, ASIPs and embedded processors are used in hand-held devices and hence generally have low power requirements. Further, because of the widespread use in video games and other consumer devices, embedded processors have low cost requirements too. As a consequence, ASIPs may result in pipelines with complex structural hazards. In such architecture, exploiting higher ILP is a major challenge to the designer. Existing techniques either schedule hardware pipelines to obtain higher throughput or perform aggressive instruction scheduling and software

pipelining [9, 16, 15, 17] to achieve higher ILP. In *modulo scheduling* or *software pipelining*, instructions from successive iterations are overlapped to exploit higher ILP. Existing techniques for exploiting ILP using software pipelining methods do not integrate the scheduling of hardware and software pipelines to achieve higher throughput and better utilization of the hardware resources. Our earlier work [4] adapts classical pipeline theory [8], to co-schedule hardware and software pipelines to obtain efficient schedules.

In this paper we develop the underlying theory called the *Modulo-Scheduled Pipeline theory* (or *MS-pipeline theory*), to co-schedule the hardware and software pipelines. More specifically,

1. We establish the necessary and sufficient condition for achieving the maximum throughput in a given pipeline operating under modulo scheduling.
2. We establish a sufficient condition to achieve a specified number of instruction initiations (less than the theoretical maximum) in a hardware pipeline operating under modulo scheduling. Also, using the sufficient condition, we provide a methodology for designing hardware pipelines that achieve the maximum throughput.

The MS-pipeline theory provides a mechanism for analyzing the throughput of MS-pipelines, and is also useful in designing pipelines to achieve a specified throughput. Thus, the MS-pipeline theory is in general useful in hardware-software co-design [7], and in particular, suitable for ASIP or embedded processor design.

The rest of the paper is organized as follows. In the following section we provide the necessary background for MS-pipeline theory. In Section 3, we establish the necessary and sufficient condition for achieving the maximum throughput in a hardware pipeline operating under modulo scheduling. Section 4 describes the design method that changes the resource usage pattern to achieve higher throughput. A comparison of related work is presented in Section 5 and concluding remarks are provided in Section 6.

2 Background and Motivation

In this section, first we present the necessary background. In Section 2.2, we review our earlier work on co-scheduling.

2.1 Background

In software pipelining, we focus on periodic schedules under which an instruction i in iteration j is initiated at time $j * \mathbf{II} + t_i$, where $\mathbf{II}$ is the *initiation interval* or period of the schedule and t_i is a constant. In other words, successive instances of an instruction i corresponding to different iterations are scheduled $\mathbf{II}$ time steps apart. The initiation interval $\mathbf{II}$ of a software pipelined schedule is governed by both resource constraints and recurrences in the loop [16, 9, 15, 3]. The minimum initiation interval $\mathbf{MII}$ is the maximum of Recurrence $\mathbf{MII}$ (**RecMII**), due to inter-iteration de-

pendences in the loop, whereas Resource **MII** (**ResMII**) is due to the resource constraints of the architecture [15].

The resource usage of an instruction (or an instruction class, in general) is represented by a two dimensional *Reservation Table* [8]. An entry X in the i th row and j th column indicates that the i th stage of the pipeline is used in j th time step by the instruction class. The distance f between two X marks in any row correspond to a *forbidden latency* [8], as initiating two instructions separated by f cycles causes a *resource conflict*, i.e., the two instructions require the usage of a pipeline stage at the same time step. Consider the resource usage of an instruction class I shown in Figure 1(a). In this example, latency 2 is forbidden.

Suppose there are two instructions i_1 and i_2 in a loop that belong to instruction class I and need to be scheduled in a pipeline. Further assume the initiation interval $\Pi = 6$ and that instructions i_1 must be scheduled at time step 0 and i_2 to be scheduled in any time step between 1 to 3 (both inclusive). As mentioned earlier, initiating instructions i_1 and i_2 with a latency 2 is forbidden. Further, with an $\Pi = 6$, the latency 4 is also forbidden, as resource usage wraps around in modulo scheduling. Hence if instruction i_1 is scheduled at time 0, then the other instruction can be scheduled at time 1 or 3. Suppose instructions i_1 and i_2 are scheduled at time steps 0 and 1 respectively. The resource usage of this pipeline represented in the *modulo reservation table* [15, 17] is as shown in Figure 1(b).

Stage	Time Steps			
	0	1	2	3
1	x		x	
2		x		x
3			x	

(a) Reservation Table

Stage	Time Steps					
	0	1	2	3	4	5
1	i_1	i_2	i_1	i_2		
2		i_1	i_2	i_1	i_2	
3			i_1	i_2		

(b) Modulo Reservation Table

Figure 1: Resource Usage in a Software Pipelined Schedule

In the above hardware pipeline and for the given Π , another instruction i_3 of the same instruction class I cannot be initiated at any time step. This can be verified by trying to initiate i_3 at time steps 4 or 5, which causes a resource conflict. In fact, even if instructions i_2 is scheduled at time 3 (or 5), it can be seen that the third instruction i_3 cannot be scheduled in the pipeline at any time step. Hence, it is obvious that at most two instructions can be initiated in the above hardware pipeline (operating under modulo scheduling) with an $\Pi = 6$.

2.2 Review of Modulo Schedule Pipelines

A formal method which analyzes hardware pipelines operating under modulo scheduling (henceforth referred to as Modulo-Scheduled Pipelines, or MS-Pipelines) and establishes the maximum

number of initiations was developed by us for the co-scheduling framework in [4, 6]. In MS-pipelines, the resource usage is represented by a modified form of the reservation table, called the *Cyclic Reservation Table (CRT)*. The CRT has exactly Π columns. If the original reservation table had fewer than Π columns, then blank columns are appended at the end to get Π columns. Otherwise, the reservation table is *folded* to Π columns; a resource usage at (i, j) (i th row and j th column) is represented at $(i, (j \bmod \Pi))$ in the CRT¹. As an example, the reservation table in Figure 1(a) yields the CRTs shown in Figure 2 for $\Pi = 3$ and $\Pi = 6$.

Stage	Time Steps		
	0	1	2
1	x		x
2	x	x	
3			x

(a) CRT for $\Pi = 3$

Stage	Time Steps					
	0	1	2	3	4	5
1	x		x			
2		x		x		
3			x			

(b) CRT for $\Pi = 6$

Figure 2: Cycle Reservation Tables

In MS-pipelines, a latency f is set to be *forbidden* if the two X marks in a row of the CRT separated by f time steps, taking into account the wrap around. For example, the X mark in (1,2) in Figure 2(b) is separated by 4 time steps from the other X mark at (1,0), taking into account the wrap-around in the CRT. It can be seen that latencies 0, 2, and 4 are forbidden for the CRT shown in Figure 2(b). The latencies that are not forbidden are *permissible*. Thus, the permissible latency set for the CRT is $\{1, 3, 5\}$. The following lemma on forbidden/permissible latencies holds in MS-pipelines [4].

Lemma 2.1 *If f is a forbidden latency in an MS-pipeline, then $\Pi - f$ is also forbidden. Equivalently, if p is a permissible latency, then $\Pi - p$ is also permissible.*

To analyze the MS-pipeline and to identify latency sequences that result in the maximum number of initiations, we construct the MS-state diagram [4] as in the case of classical pipeline theory. Each state in the state diagram is represented by the set of permissible latencies in that state. The initial state in the MS-state diagram represents an initiation at time 0 and is associated with the initial permissible latency set. The following steps are involved in the construction of the state diagram.

Procedure 1 Construction of State Diagram:

Step 1 Start with the initial permissible latency set S_0 for the initial state.

¹ With this folding, X marks separated by Π columns may be placed on the same column in the CRT. However, such Π 's are not permitted under modulo scheduling constraint.

Step 2 For each permissible latency p_i in the current state S , derive a successor state S' with the following permissible latency set.

$$S' = S_{-p_i} \cap S_0 \quad \text{where} \quad S_{-p_i} = \{(p_j - p_i) \bmod \mathbf{II} \mid \forall p_j \in S.\}$$

(See explanation below.)

The set S_{-p_i} is obtained by subtracting p_i , the chosen latency, from each permissible latency p_j in S . The subtractions are performed modulo $\mathbf{II}$. The resulting set is the set of *possible* permissible latencies at the current state. Of these latencies, those belonging to the initial forbidden latency set are still forbidden; other are permissible. Thus the intersection with the initial permissible latency set yields the set of permissible latencies in the new state S' .

Figure 3: Modulo-Scheduled State Diagram

The MS-state diagram for the example CRT of Figure 2(b) is shown in Figure 3(a). A path $S_0 \xrightarrow{p_1} S_1 \xrightarrow{p_2} S_2 \cdots \xrightarrow{p_k} S_k$ in the MS-state diagram corresponds to k initiations that are made at time steps $0, p_1, (p_1 + p_2), \cdots, (p_1 + p_2 + \cdots + p_k)$. It can be seen that at most two initiations can be achieved in the MS-pipeline as all paths are between S_0 to S_1 with a latency of 1, 3, or 5.

3 Analysis of MS-Pipelines

In this section first we motivate the need for MS-pipeline theory. We develop the MS-pipeline theory in the subsequent subsections.

3.1 Motivation for MS-Pipeline Theory

In the CRT of Figure 2(b), any stage of the pipeline is used for at most 2 time steps. Hence, in an $\mathbf{II}$ of 6 cycles, not more than $\lfloor \frac{6}{2} \rfloor = 3$ instructions can be initiated in the pipeline. We refer to this informally as the theoretical upper bound on the number of initiations. However, our earlier discussions in Sections 2.1 and 2.2 indicate that only 2 instructions can be initiated in the given pipeline. This raises the question, can the number of instructions initiated in a pipeline be improved to its theoretical upper bound, in this example to 3. As the designer of the micro-architecture of the

pipeline, if the resource usage pattern of the pipeline can be modified (keeping the resources usages intact), then it should be possible to achieve 3 instruction initiations in the given pipeline. More specifically, assume that we introduce a delay (or dummy stage) at time step 2 in row 1 (represented by d in Figure 4(a)). This is known as delay insertion in the classical pipeline theory [11]. The resource usage of the pipeline changes as shown in Figure 4(a). The permissible latency set for the modified CRT is $\{1, 2, 4, 5\}$. Now it is possible to have 3 initiations, represented by x, y, and z at time steps 0, 2 and 4 respectively (refer to Figure 4(b)).

Stage	Time Steps					
	0	1	2	3	4	5
1	x		d	x		
2		x			x	
3				x		

(a)

Stage	Time Steps					
	0	1	2	3	4	5
1	x	z	y	x	z	y
2	y	x	z	y	x	z
3		z		x		y

(b)

Figure 4: Modified Reservation Table Supporting 3 Initiations

3.2 Bound on Maximum Initiations

The number of initiations possible in a pipeline is bounded by two factors, UB_perm and UB_res [4].

Theorem 3.1 [GovindarajanAltmanGao [4]] *The upper bound (henceforth referred to as UB_Init) on the number of initiations made in an MS-pipeline during $\mathbf{II}$ cycles is*

$$UB\_Init = \max(UB\_res, UB\_perm), \text{ where } UB\_res = \left\lfloor \frac{\mathbf{II}}{d_{max}} \right\rfloor, UB\_perm = (k + 1),$$

k is the cardinality of the permissible latency set and d_{max} is the maximum number of x marks in any row in the reservation table.

By definition, instructions can be initiated only on a permissible latency. Further, at most one instruction can be initiated at each permissible latency; another instruction initiated at this time step violates the modulo scheduling constraint and thereby causes structural hazards at every stage. Hence the value of UB_perm should be one more than the number of permissible latencies. The second bound UB_res is due to resource usage, and equals $\lfloor \mathbf{II}/d_{max} \rfloor$.

It is important to observe here that the maximum number of initiations (referred to as Max_Init) computed from the longest path in the MS-state diagram is what is actually achievable in the MS-pipeline. The bound on UB_Init due to Theorem 3.1 is such that $Max\_Init \leq UB\_Init$.

3.3 Necessary and Sufficient Condition for Achieving UB_Init

In this section we derive the necessary condition for Max_Init to equal the UB_Init . First, we define a special form arithmetic progression which helps characterize the performance of MS-pipelines.

Definition 3.1 An arithmetic progression (AP) of the form $p, 2p, \dots, k.p$ is called a **special form AP**.

For example the sequence $(3, 6, 9)$ is a special form AP. Suppose, the initial permissible latency set is a special form AP, e.g., $S_0 = \{3, 6, 9\}$ and $\mathbf{II} = 12$. Further assume $UB_{perm} \leq UB_{res}$. Thus, by Theorem 3.1, the UB_{Init} is $|S_0| + 1 = 4$. It can be seen that there exists a path $S_0 \xrightarrow{3} S_1 \xrightarrow{3} S_2 \xrightarrow{3} S_3$ in the MS-state diagram which corresponds to 4 initiations at time steps 0, 3, 6, and 9. The following theorem formalizes this idea.

Theorem 3.2 Consider an MS-pipeline with a modulo initiation interval $\mathbf{II}$. Let the cardinality of the initial permissible latency set be k where k is less than $\lfloor \mathbf{II}/d_{max} \rfloor$. Then $(k + 1)$ initiations are possible in this pipeline if and only if the initial permissible latency set forms a special form AP.

The reader is referred to [5] for a formal. Theorem 3.2 is a strong result. However its applicability is restricted to cases where the permissible latency set is a special form AP. In the following section we present a weaker, but more useful, result which specifies a sufficient condition for achieving a given number of initiations.

3.4 Sufficient Condition

Consider the initial permissible latency set $\{1, 2, 4, 5\}$ of the modified CRT of Figure 4(a). It contains a subsequence $\{1, 2\}$ and $\{2, 4\}$ which are special form APs. The MS-state diagram contains paths $S_0 \xrightarrow{1} S_1 \xrightarrow{1} S_2$ and $S_0 \xrightarrow{2} S_3 \xrightarrow{2} S_2$ which correspond to 3 initiations. Thus when there is a subsequence (which is a special form AP) of length l , then at least $l + 1$ initiations are possible. The following theorem formalizes this notion as a sufficiency condition.

Theorem 3.3 If the initial permissible latency set S_0 contains a special form arithmetic progression of length l , then at least $(l + 1)$ initiations are possible in the MS-pipeline.

4 Improving MS-Pipeline Performance via Delay Insertion

In this section we develop a technique to modify the CRT in order to realize UB_{Init} initiations. It is important to note here that UB_{perm} no longer plays a role in determining UB_{Init} , as the permissible latency set itself changes with delay insertion.

4.1 Delay Insertion to Improve Number of Initiations

Let us once again consider the CRT from our motivating example (Figure 2(b)). The Max_{Init} achievable in this CRT (as shown by the MS-state diagram in Figure 3) is 2. However, the UB_{Init} for this CRT (as discussed earlier) is 3. Now, in order to have $Max_{Init} = UB_{Init}$, by Theorem 3.3,

the permissible latency set should be adjusted² to contain a special form AP of length $UB\_Init - 1$. Thus, in our example, the permissible latency set should contain a special form AP of length 2. Let the AP be $\{p, 2p\}$. Clearly, $2.p < \mathbf{II}$; i.e., $p < \mathbf{II}/2$; or $p < 3$. For example when $p = 1$, let us try to include the AP $P_i = \{1, 2\}$ in the permissible latency set. If the permissible latency set (denoted by P) has to include the elements 1 and 2 then, according to Lemma 2.1, it will also include their complements $\mathbf{II} - 1 = 5$ and $\mathbf{II} - 2 = 4$. Thus $P = \{1, 2, 4, 5\}$ is a permissible latency set that will guarantee at least 3 initiations. From this, the forbidden latency set F is $\{0, 3\}$.

The next step is to adjust the CRT such that its cyclic forbidden latency set is $\{0, 3\}$. Patel and Davidson [11] discuss how to construct a reservation table for a given forbidden latency set. We observe that this approach is directly applicable for MS-pipelines as well. A detailed discussion on this can be found in [5]. The delay insertion method inserts a delay at (1,2) and the resulting CRT is same as in Figure 4(a).

The introduction of delays in an MS-pipeline may increase the execution time of an operation; this, in turn, will affect Recurrence $\mathbf{MII}$ (**RecMII**) and $\mathbf{II}$. Hence our delay insertion method is applicable only to cases where either there are no loop-carried dependences or **ResMII** dominates. However, it was observed in [17] that in a large majority of loops **ResMII** dominates **RecMII** and hence governs $\mathbf{MII}$.

4.2 Achieving the Upper Bound on Maximum Initiations

One relevant question in this context is: Does the delay insertion method always succeed? We answer this question affirmatively by establishing the following theorem.

Theorem 4.1 *Given an initiation interval $\mathbf{II}$, a CRT with at most d_{max} X marks on each row, it is always possible to achieve $u = \lfloor \mathbf{II}/d_{max} \rfloor$ initiations by introducing delays in the CRT.*

A formal proof based on constructing a modified CRT is presented in [5]. In the modified CRT, the d_{max} X marks are placed at cycles $0, u, 2.u, \dots, (d_{max} - 1).u$, which supports the permissible latency set $\{1, 2, \dots, (u - 1)\}$.

5 Related Work

The classical pipeline theory was developed to improve the throughput and the utilization of pipelined and vector architectures [8, 11]. It has also been applied to cyclic job-shop scheduling in manufacturing systems [2]. The theory of MS-Pipelines is different from that of classical pipelines in that the period of the latter is completely determined by the pipeline structure. Whereas in the case of MS-pipelines, the initiation interval depends on other factors, such as recurrences in the scheduled loop and resource availability, in addition to the pipeline structure. Recently,

---

²As Theorem 3.3 is only a sufficient condition, it may be possible to achieve 3 initiations without S_0 containing a special form AP of length 2. We do not consider such alternative possibilities in this paper.

ideas from hardware pipeline theory have been used to develop a Finite State Automaton (FSA)-based approach to instruction scheduling [1, 10, 13, 14]. However, these techniques are applied to instruction scheduling, while our method focuses on software pipelining. Several examples are given in [4, 6] to illustrate that neither classical pipeline theory nor the FSA-based approaches can directly be used to analyze pipeline structures operating under modulo scheduling.

In this paper we have developed the MS-pipeline theory for architectures in which the processors do not share any resources. However, the MS-pipeline theory and co-scheduling have been extended in [21] for pipelines that share resources; *e.g.*, the normalization stage being shared by the floating point add and floating point multiply pipelines. In such cases the size of the MS-state diagram could grow extremely large even for moderate values of II . In [6, 21], (i) a mechanism for identifying distinct paths and (ii) alternative efficient approaches to construct the MS-state diagram have been proposed. These results make co-scheduling a viable approach for analyzing and designing pipelines for ASIPs and embedded processors.

6 Conclusions

In this paper, we have developed the theory of MS-pipelines. We have analyzed MS-pipelines to choose latency sequences that improve the utilization of the MS-pipeline. We have established the necessary and sufficient condition to achieve the upper bound (UB_{Init}) on the number of initiations. Using the proposed MS-pipeline theory, we have developed a delay insertion method to design pipelines that achieve maximum throughput. This design methodology is extremely useful in designing ASIPs and embedded processors that exploit higher ILP. The proposed theory and the delay insertion methods are also useful in the context of reconfigurable architectures [19, 20] for exploiting greater ILP.

References

- [1] V. Bala and N. Rubin. Efficient instruction scheduling using finite state automata. In *Proc. of the 28th Ann. Intl. Symp. on Microarchitecture*, pages 46–56, Ann Arbor, MI, Nov. 29–Dec.1, 1995.
- [2] J.K. Chaar and E.S. Davidson. Cyclic job shop scheduling using collision vectors. Technical Report CSE-TR-169-93, University of Michigan, Ann Arbor, MI, Aug. 1993.
- [3] J. C. Dehnert and R. A. Towle. Compiling for Cydra 5. *J. of Supercomputing*, 7:181–227, May 1993.
- [4] R. Govindarajan, Erik R. Altman, and Guang R. Gao. Co-scheduling hardware and software pipelines. In *Proc. of the Second Intl. Symp. on High-Performance Computer Architecture*, pages 52–61, San Jose, CA, Feb. 1996.
- [5] R. Govindarajan, E. R. Altman, and G. R. Gao. Modulo-Scheduled Pipelines. ACAPS technical memo, School of Computer Science, McGill University, Montréal, Québec, June 1996.

- [6] R. Govindarajan, N.S.S. Narasimha Rao, E.R. Altman, and G.R. Gao. An enhanced Co-Scheduling method using reduced MS-State diagram. In *Proc. of the 12th International Parallel Processing Symposium and 9th Symposium on Parallel and Distributed Processing*, Orlando, FL, March 1998.
- [7] R. K. Gupta and G. De Micheli. Hardware-Software Cosynthesis for Digital Systems. *IEEE Design & Test of Computers*, pages 29–41, Sept. 1993.
- [8] P. M. Kogge. *The Architecture of Pipelined Computers*. McGraw-Hill Book Co., New York, NY, 1981.
- [9] M. Lam. Software pipelining: An effective scheduling technique for VLIW machines. In *Proc. of the SIGPLAN '88 Conf. on Programming Language Design and Implementation*, pages 318–328, Atlanta, GA, Jun. 1988.
- [10] T. Muller. Employing finite state automata for resource scheduling. In *Proc. of the 26th Ann. Intl. Symp. on Microarchitecture*, Austin, TX, Dec. 1993.
- [11] J. H. Patel and E. S. Davidson. Improving the throughput of a pipeline by insertion of delays. In *Proc. of the 3rd Ann. Symp. on Computer Architecture*, pages 159–164, Clearwater, FL, Jan. 1976.
- [12] Philips Inc. *TriMedia Processor*, 1995.
- [13] T. A. Proebsting and C. W. Fraser. Detecting pipeline structural hazards quickly. In *Conf. Rec. of the 21st ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, pages 280–286, Portland, OR, Jan. 1994.
- [14] V. Janaki Ramanan and R. Govindarajan. Resource usage models for instruction scheduling: Two new models and a classification. In *Proc. of the Intl. Conf. on Supercomputing*, Rhodes, Greece, 1999.
- [15] B. R. Rau and J. A. Fisher. Instruction-level parallel processing: History, overview and perspective. *J. of Supercomputing*, 7:9–50, May 1993.
- [16] B. R. Rau and C. D. Glaeser. Some scheduling techniques and an easily schedulable horizontal architecture for high performance scientific computing. In *Proc. of the 14th Ann. Microprogramming Work.*, pages 183–198, Chatham, MA, Oct. 1981.
- [17] B. R. Rau. Iterative modulo scheduling: An algorithm for software pipelining loops. In *Proc. of the 27th Ann. Intl. Symp. on Microarchitecture*, pages 63–74, San Jose, CA, Nov. 30–Dec.2, 1994.
- [18] Texas Instruments. *TMS-320C6x Users Guide*, 1997.
- [19] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal. Barring it to all software: Raw Machines. *IEEE Computer*, 30(9):86–93, Sept. 1997.
- [20] M. Weinhardt. Compilation and pipeline synthesis for reconfigurable architectures. loops. In *Reconfigurable Architectures – High Performance by Configware (Proc. of the RAW'97)*, April 1997.
- [21] C. Zhang, R. Govindarajan, S. Ryan, and G.R. Gao. Efficient state-diagram construction methods for software pipelining., In *Proc. of the Compiler Construction Conference*, Amsterdam, The Netherlands, March 1999.